mysqljsonload 1.1

JSON file loader for MySQL

mysqljsonload 1.1
2012-06-14
Anders Karlsson, Recorded Future AB
anders@recordedfuture.com

Table of contents

TABLE OF CONTENTS **2**
INTRODUCTION **2**
BUILDING MYSQLJSONLOAD **3**
CONFIGURING AND USING MYSQLJSONLOAD **4**
MYSQLJSONLOAD CONFIGURATION OPTIONS REFERENCE **5**
CHANGE LOG **6**
VERSION 1.1 6
VERSION 1.0 7

Introduction

MySQL has come a long way and when this document is written, in May 2012, MySQL 5.6 is around the corner. At the same time, there is a lot of competition, the different NoSQL alternatives are breaking new ground where MySQL used to be king. In reality, what has opened up is not only a market with several alternatives that are much less common than what used to be the case (i.e. “Which SQL database should I choose”) but an environment where these alternatives coexist.

In addition to this, there is now also many more data formats, partly because of these new technologies (which many, me included, doesn’t really consider “new” in terms of not having existed before), and partly because we deal with new kinds of data, like GPS positioning data, document based data, lists of items, objects etc.

One of the many new formats of data that is appearing is JSON (JavaScript Object Notation). JSON has proved to be very popular not only with the JavaScript focused technologies, but as a generic format used in conjunction with REST and as a data transformation format.

In the case of MySQL, there isn’t really that much JSON support. Instead, MySQL seems to rely on CSV (Coma Separated Values) for data exchange. CSV is in some ways a neat format, but it is not standardized and has it’s roots in table/column/field only world. We now have objects, we have to understand that, and we have unstructured data. This fits easily in JSON and MySQL can sure handle it, but there are few, or any tools to support this. So along comes **mysqljsonload**. This is a tool that allows you to load a file with JSON based data to be loaded into MySQL. It is not a very advanced tool, but at least it understands JSON and MySQL and does some basic mappings, with the intention that data is then massaged further by standard SQL.

The program is a command line based tool, and relies on *jansson* for JSON processing. Beyond that, the loading process is multi-threaded and supports MySQL array inserts, and has a bunch of configuration options, which may be used in a configuration file, on the command line, or a combination of the two.

Building mysqljsonload

mysqljsonload use the *autobuild* tools for building, so building it follows these steps:

- Download **mysqljsonload** package from sourceforge: https://sourceforge.net/projects/mysqljson
- Unpack it
- Configure it
- Make it
- Optionally install it

Like this example:

```
$ wget http://sourceforge.net/projects/mysqljson/files/myjsonload_1.1/mysqljsonload-
      1.1.tar.gz
$ tar xvfz mysqljsonload-1.1.tar.gz
$ cd mysqljsonload-1.1
$ ./configure -with-mysql=/usr/local/mysql
$ make
```

Following this, the program is available in the directory where you built it. You can install it to the location specified with the optional *--prefix* (the installation will copy the program to the *bin* subdirectory of this location) option to *configure* (the default is */usr*). If you install to /usr you need root privileges of course.

```
$ sudo make install
```

Once the program is built, we are ready to configure it. You can either use a normal MySQL configuration file, pass the options on a command line, or use a combination of the two. For normal MySQL connection options, like socket, port and username, these are also read from the *client* section MySQL usual configuration files. See mysqljsonload configuration options reference for a reference to all options.

If you have problems building mysqljsonload, the most likely issues are:

- The MySQL libraries aren't found. MySQL must be installed and if not in any of the standard places, then pass the *--with-mysql* option.
- The *Jansson* library isn't found. This library is used for JSON parsing and is needed by *mysqljson*. Look at the Jansson website (http://www.digip.org/jansson) for instructions of where to download it and how to build it.

If you install the *jansson* library in a non-standard location, for example if you do not have privileges to install it into */usr/* or any of the other standard locations, then use the *–with-jansson* option when configuring, to point to the appropriate location. The same is true for MySQL. For example if you

installed the jansson library in */home/ken/jansson*, and with MySQL installed in */usr/local/mysql565*, then the configure command would look like this:

```
$ ./configure --with-mysql=/usr/local/mysql565 --with-jansson=/home/ken/jansson
```

Configuring and using mysqljsonload

The default mysqljsonload configuration file is *jsonload.cnf* and the options are in the *jsonload* section in this file. The required options are a database name and a table name to load into and the name of a file to load.

One you have the configuration file set up, you can start the program, and you probably want it running in the background:

```
£ ./mysqljsonload --database test --table mydata mydata.json
```

The way the loading works is that the columns in the table that is loaded into determines which attributes of the JSON objects to load. This comparison is case insensitive. A simple example table would be this:

```
CREATE TABLE users(id INTEGER NOT NULL PRIMARY KEY,
  firstname VARCHAR(256) NOT NULL,
  middlename VARCHAR(256),
  lastname VARCHAR(256) NOT NULL,
  user_group INT NOT NULL,
  active BOOL NOT NULL);
```

If we then have a JSON file, *users.json*, with the following contents:

```
{"_id": 1, "firstname": "John", "middlename": "Frank", "lastname": "Doe", "interests":
       ["cars", "books"], "active": false}
{"_id": 2, "firstname": "Anne", "middlename": null, "lastname": "Doe", "interests":
       ["diving"], "active": true}
```

We have three possible issues:

- The *id* column isn't present in the JSON file, instead this column is called *_id*.
- The value for the *group* column isn't present in the JSON file, but it is a NOT NULL column, so we need to provide a value. We know the group is 57.
- The *interests* column is a JSON array and isn't present.

To fix this, we have to map the column name for the *id* column. The second issue may be ignored, as the *interests* column doesn't exist in the table, it will not be loaded. We may decide to load it into a new column, but then we will have the array as loaded as a string into that column. If we decide to ignore the *interests* column, the command line might look like this:

```
mysqljsonload --col-value=group=57 --col-name-map=_id=id --table=users --database=test users.json
```

If you want to use a configuration file instead, then this would look like this:

```
[jsonload]
database=test
table=users
```

```
col-value=group=57
col-name-map=_id=id
file=users.json
```

mysqljsonload configuration options reference

The mysqljsonload program takes many options, and is very flexible. The options may either be in a configuration file or passed on the command line or both, the options then prefixed with double dashes (--). Some options have single character shortcuts, then they are prefixed with a single dash. When used in the configuration file, only the long option may be used.

In the configuration file, defined by the *–defaults-file* option, the options must be in the *jsonload* section. The *client* section in the normal MySQL configuration files is also read, to pick to defaults for the MySQL host, socket and such things.

- array-strip – Skip leading and trailing brackets when loading an array into a column.
- array-to-null – Handle an array in the data being loaded as NULL, the default is to load it as a string.
- bool-as-int – Handle a JSON bool as an int instead of the usual MySQL TRUE / FALSE values.
- col-default – Specify column default values (this option ay appear more than once). If a column to load isn't included in the data being loaded, then it is instead loaded with this value. The default is to set undefined columns to NULL. The format of this option is <column name>=<value>, for example:
  --col-default=col1=57
- col-value – Set a named column in the database to a fixed value, you may have more than one of these options. The format is <column name>=<value> to to set all occurences of the column *group* to 57, use:
  --col-value=group=57
- col-name-map – Mapping of column names in JSON to an alternate name of MySQL. This option may be specified several times, and has the format <name>=>alternate name>. For example *–col-name-map=id=_id*
- commit-interval – The number of .records per commit in a thread. This is only checked when an array is to be inserted, so more than this # of records may be inserted. If set to 0 or 1, autocommit is used.
- database / d – Name of database to use. This is a required option.
- defaults-file – Name of configuration file to use.
- empty-null – Treat an empty string, array or object as NULL.
- file – Name of file to load. Alternatively you can just pass the filename of the commandline. If there is a filename in the configuration file, it will be overridden by an option on the commandline.
- full-status – Flag to show full status information after loading.
- help – Show help. This option can only be entered on the commandline.
- host – The hostname of the MySQL server. Default is NULL (localhost).

- insert-array-size – Number of records per array to insert.
- logfile – File for log messages. Default is standard error.
- loglevel – Level of logging. Valid values are status, error, info, verbose or debug. The default level is info.
- log-interval - # of records between progress records being written to the log file. The default is 0 which means that no progress records are written at all.
- null-string – An alternative way of representing NULL JOSN values in MySQL, the default is to map a JSON null to a MySQL NULL.
- nullcol – Set the specified column to NULL (this option ay appear more than once).
- object-to-null – Load embedded objects in JSON as NULL to the database.
- password / p – Password for user to connect to the database with.
- port / P – MySQL Server port to connect to.
- progress – Number of input file records to process between showing a progress indicator.
- prtrec – Print records as they are loaded.
- skipcol – Skip the specified column (this option ay appear more than once).
- skip-lines – Skip the specified number of lines before starting to load data.
- skip-json-records – Skip the specified number of JSON records before starting to load.
- socket / S – The MySQL Server socket to connect to.
- sql-init – SQL statements to run before starting to load. You may specify many of these.
- sql-retry – The number of times to retry a failed MySQL INSERT statement.
- sql-retry-sleep – The number of seconds to sleep between retrying an SQL statement.
- stop-on-error – Stop if there is an error when executing a clean-sql statement. This is the default.
- stop-on-init-error – Stop if one of the sql-init statements fails.
- table – Name of the table to load into. This is a required option.
- threads / t – Number of clean-sql threads to run. Note that you may have more threads that clean-SQL statement, or less. The default is 5 threads.
- user / u – MySQL user to connect to the database with.
- utf8 – Enable UTF-8 mode. This option is on by default.
- version – Show version. This is a command-line only option.

Change log

Version 1.1

This is mainly a feature release, which also has a few bugfixes.

- Fixed bug in the way commit and autocommit is handled.
- Added retry of failed SQL statements.
- Logging is improved with some thread specific data being printed.
- Added skip on leading JSON records, in addition to skipping lines.
- Added handling or true, false and null JSON values.

- Added sql-init and supporting stop-on-init-error options.

Version 1.0

This is the first public release.